Звіт
Лабораторна робота 4
ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ
Большаков Андрій МІТ-31
https://github.com/Utilka/OOP_labs_Univ

ІНТЕРФЕЙСИ ТА ПРОТОКОЛИ У РУТНОК

Мета: розглянути поняття інтерфейсу та його призначення. Розглянути можливості Пайтон по створенню і використанню інтерфейсів.

main.py

```python
import abc
from abc import ABCMeta
from random import randrange
from typing import Union, runtime_checkable, Tuple, Protocol

from Crypto.Cipher import AES
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.PublicKey.RSA import RsaKey
from Crypto.Random import get_random_bytes

Buffer = Union[bytes, bytearray, memoryview]

@runtime_checkable
class MyByteCipher(Protocol):

    def encrypt(self, clear_message: bytes) -> bytes:
        pass

    def decrypt(self, encrypted_message: bytes) -> bytes:
        pass


class MyCipher(metaclass=abc.ABCMeta):

    @abc.abstractmethod
    def encrypt(self, clear_message):
        pass

    @abc.abstractmethod
    def decrypt(self, encrypted_message):
        pass


class MySymmetric(MyCipher, metaclass=ABCMeta):

    def __init__(self, private_key):
        self._private_key = private_key


class MyAsymmetric(MyCipher, metaclass=ABCMeta):

    def __init__(self, private_key, public_key):
```

```
        self._public_key = public_key
        self._private_key = private_key


class MyAES(MySymmetric):

    def __init__(self, private_key: bytes = None):
        if (private_key is None):
            private_key = self.generate_key()

        super().__init__(private_key)

    @staticmethod
    def generate_key(length=16):
        key: bytes = get_random_bytes(length)
        # key = b'Sixteen byte key'
        return key

    def encrypt(self, clear_message: Buffer):

        cipher = AES.new(self._private_key, AES.MODE_EAX)
        nonce: bytes = cipher.nonce
        ciphertext, tag = cipher.encrypt_and_digest(clear_message)
        return (nonce, ciphertext, tag)

    def decrypt(self, encrypted_message: Tuple[bytes, bytes, bytes]):
        """

        :type encrypted_message: tuple (nonce,ciphertext,tag)
        """
        nonce, ciphertext, tag = encrypted_message

        cipher = AES.new(self._private_key, AES.MODE_EAX, nonce=nonce)
        plaintext = cipher.decrypt(ciphertext)
        try:
            cipher.verify(tag)
            return plaintext
        except ValueError:
            print("Key incorrect or message corrupted")

class MyRSA(MyAsymmetric):

    def __init__(self, private_key: RsaKey = None, public_key: RsaKey = None):
        if (private_key is None):
            private_key = self.generate_key()
        if (public_key is None):
            public_key = private_key.publickey()

        super().__init__(private_key, public_key)

    @staticmethod
    def generate_key(length: int = 4096):
        private_key: RsaKey = RSA.generate(length)
        return private_key

    def encrypt(self, clear_message: Buffer):
        encryptor = PKCS1_OAEP.new(self._public_key)
        encrypted = encryptor.encrypt(clear_message)
        return encrypted
```

```
    def decrypt(self, encrypted_message: Buffer):
        decrypter = PKCS1_OAEP.new(self._private_key)
        decrypted = decrypter.decrypt(encrypted_message)
        return decrypted


class MyVigenere(MySymmetric):
    def __init__(self, private_key: str = None):
        if (private_key is None):
            private_key = self.generate_key()

        super().__init__(private_key)

    @staticmethod
    def generate_key(length: int = 10):
        private_key = "".join([chr(randrange(26) + 65) for i in range(length)])
        return private_key

    def _v_cypher(self, text: str, enc: bool):
        key = self._private_key
        if isinstance(text, bytes):
            text = text.decode("utf-8")
        text = text.upper()
        key_length = len(key)
        key_as_int = [ord(i) for i in key]
        text_as_int = [ord(i) for i in text]
        if enc:
            ciphertext = ''
            for i in range(len(text_as_int)):
                value = (text_as_int[i] + key_as_int[i % key_length]) % 26
                ciphertext += chr(value + 65)
            return ciphertext
        else:
            plaintext = ''
            for i in range(len(text_as_int)):
                value = (text_as_int[i] - key_as_int[i % key_length]) % 26
                plaintext += chr(value + 65)
            return plaintext

    def encrypt(self, plaintext: str) -> str:
        return self._v_cypher(plaintext, True)

    def decrypt(self, ciphertext: str) -> str:
        return self._v_cypher(ciphertext, False)


if __name__ == '__main__':
    my_aes = MyAES()
    enc_m = my_aes.encrypt(b"someMyAEStext")
    dec_m = my_aes.decrypt(enc_m)
    print(dec_m)

    my_rsa = MyRSA()
    enc_m = my_rsa.encrypt(b"someMyRSAtext")
    dec_m = my_rsa.decrypt(enc_m)
    print(dec_m)

    my_vig = MyVigenere()
    enc_m = my_vig.encrypt("someVigenetext")
```

```
    dec_m = my_vig.decrypt(enc_m)
    print(dec_m)

    print("isinstance(my_aes,MyByteCipher):",isinstance(my_aes,MyByteCipher))
    print("isinstance(my_rsa,MyByteCipher):",isinstance(my_rsa,MyByteCipher))
    print("isinstance(my_vig,MyByteCipher):",isinstance(my_vig,MyByteCipher))
```

collection.py

```
from collections.abc import Collection
from main import MyCipher

from typing import Iterator, Iterable, Optional


class MyCipherColl(Collection):

    def __init__(self, cypher_list: Iterable[MyCipher]):
        for item in cypher_list:
            if not isinstance(item, MyCipher):
                raise TypeError("all objects in collection must be of type MyCipher")
        self._list = list(cypher_list)

    def __contains__(self, item: object) -> bool:
        return self._list.__contains__(item)

    def __iter__(self) -> Iterator[MyCipher]:
        return self._list.__iter__()

    def __len__(self) -> int:
        return self._list.__len__()


if __name__ == '__main__':
    import main

    c = MyCipherColl([main.MyAES(), main.MyRSA(), main.MyVigenere(), main.MyVigenere()])
    for i in c:
        enc = i.encrypt(b"Mytext")
        dec = i.decrypt(enc)
        print(enc)
        print(dec)

    for i in c:
        enc = i.encrypt(b"Mytext")
        dec = i.decrypt(enc)
        print(enc)
        print(dec)
```

из результатов прогонки основного файла видим такой результат

isinstance(my_aes,MyByteCipher): True
isinstance(my_rsa,MyByteCipher): True

isinstance(my_vig,MyByteCipher): True

из чего делаем вывод что протокол проверяет лишь наличие методов, и не проверяет типы данных указаные в анотациях

Висновок: я розглянув поняття інтерфейсу та його призначення. Розглянув можливості Пайтон по створенню і використанню інтерфейсів.